PCA Configuration Monitor: Measured and Predicted Configuration Design Document

This document illustrates the design of the configuration monitoring software of the PCA instrument.

PCA-CON-DES-V3.3

XTE-PCA Software Module

October 14th 1994

by

Vikram Savkoor

Hughes STX Corp.

4400 Forbes Blvd., Lanham, MD 20706

Foreword

The PCA science group is responsible for providing to the SOC an agreed set of software products together with their associated documentation and test data/results. Such deliveries are governed by the details to be found in the PCA IAD and ICD documents.

In order to facilitate the production of these deliverables a series of documents have been produced that are intended to contain as much relevant and accurate information as is available that pertains to the title and subject matter of each delivery item. These documents should also present and explain the concepts and plans for the use of the supplied information.

This document is concerned with the topic of PCA CONFIGURATION MONITOR.

The PCA group will also produce additional software for its own use, in its own external environment. This software will also be provided to the SOC but is not governed by such rigorous product assurance requirements although it also requires documentation.

1.0 Introduction

1.1 Identification of Document

The PCA team at GSFC is required to deliver a number of software modules to the Science Operations Center (SOC) for the XTE mission. This document attempts to examine and pull together all of the aspects relating to the development of the PCA Configuration Monitor software.

2.0 Related Documentation

2.1 Parent Documents

Not applicable to this document

2.2 Applicable Documents

There are a number of relevant documents to the general areas of configuration monitoring and verification.

Interface Control Document between the XTE Science Operations Center and the PCA Principal Investigator Team. XTE-SOC-PCA-ICD-V1.0. Rahim Choudhary et al. Provisional Draft August 1993.

PCA Commands - Types, Sequences, Use, Configurations, Screen Functions, Verification. SOC Interface. XTE-PCA Software Module, ver 2.0, Barry Giles, September 1993.

Command Generation. Leon Herreid, August 1993.

Mission Monitoring. Paritosh Malaviya and Leon Herreid. Provisional Draft August 1993.

Relevant information to the general areas and detailed format of PCA Housekeeping data can be found in the following document:

PC Housekeeping-Status, Suggested Screen Layout, Recall Modes, SOC interface, XTE-PCA Software Module, ver 1.0, Barry Giles, PCA Team, March 16, 1993

3.0 Requirements

From the SOF Responsibilities and Requirements document, there are no specific requirements regarding PCA Instrument Measured Configuration and Predicted Configuration. But there is at aleast an implicit requirement to provide the subclasses of SOC developed base classes and to implement their virtual methods.

4.0 Operating Scenario

4.1 Measured Configuration

At all times the measured configuration will reflect the latest known state of all the commandable PCA parameters. Housekeeping data from the PCA will arrive in the SOC at a semi-regular rate (all PCUs at every 8 seconds), with exceptions where in there may be some occasional gaps in time. Each incoming PCU housekeeping packet will be decoded and the commandable parameters stripped out to form an updated Measured Configuration. The SOC will archive all changes in the measured configuration, i.e. during long periods of unchanging conditions it will not be archived.

The PCA Instrument Configuration Monitoring is based on the notion of configurations. A configuration is the status of every commandable parameter of the PCA instrument electronics. The Mission Monitoring subsystem in SOC will use the measured configuration to verify that the PCA is always in the intended state it is commanded and expected to be in by comparing measured configuration values with predicted configuration values. Error conditions will result in messages and/or alarms to Operations and the Duty Scientist and will be maintained in a significant event log file. This event log may also be augmented by the Duty Scientist via manual entries.

Normally only the latest Measured Configuration entry will be used for verification purposes.

4.2 Predicted Configuration

When constructing commands the scenario is to move the instrument from one configuration (the current one) to another configuration (the desired configuration). A series of sequenced commands forms a series of predicted configurations for a series of *time marks* in the future.

A new predicted configuration is formed by merging the previous predicted configuration with the commanded changes extracted from the currrent desired. A specific future predicted configuration therefore is the resultant summation of a whole series of changes starting from some definitive known situation. Such an event or Master Reset could consist of commanding all the parameters to a specific desired configuration and resetting the predicted configuration to match. An equivalent situation occurs following power up or the sending of a RIF Reset command.

4.2.1 Configuration *feed forward* Requirement

Previous predicted configurations from the past are continuously *fed forward* in time and merged with desired configuration to form a new predicted configuration. If an EOR command is sent then it will cause a verification failure for all future predicted configuration since they were created without knowledge of the change just made.

Therefore every single EOR sent up to the PCA must also be *fed forward* to update all future predicted configuration that exist.

4.2.2 Master Resets

Previous predicted configurations from the past are continuously *fed forward* in time and merged with desired configuration to form a new predicted configuration. Any error therefore become *frozen in* requiring occasional reset operations. There are 3 basic means of doing this:

- commanding ALL the possible parameters (for all PCUs) to a desired configuration and resetting the predicted configuration to match.
- sending the RIF Reset command to all PCUs and resetting the predicted configuration to the default power up settings.
- setting the predicted configuration to a default power up state following experiment switch on

TBD - how often will a full command reset be required

4.2.3 Verification Process and Results

The SOC Mission Monitoring (MM) subsystem will keep track of the MET and the Predicted Configuration Time Line. When a new predicted state is expected a PCA supplied C++ method will compare the incoming Measured Configuration to check that the change has occurred. If the comparison reveals significant differences, a verificationReport will be generated describing the verification results.

5.0 Open issues

- What is the procedure to compare two measured configurations? How do we define the comparison results? (build 2)
- What is the verification process? How do we define the comparison results? (build 3)
- What is the procedure to handle missing packets and packets received out of time-order sequence?

6.0 Design Features

6.1 Measured Configuration

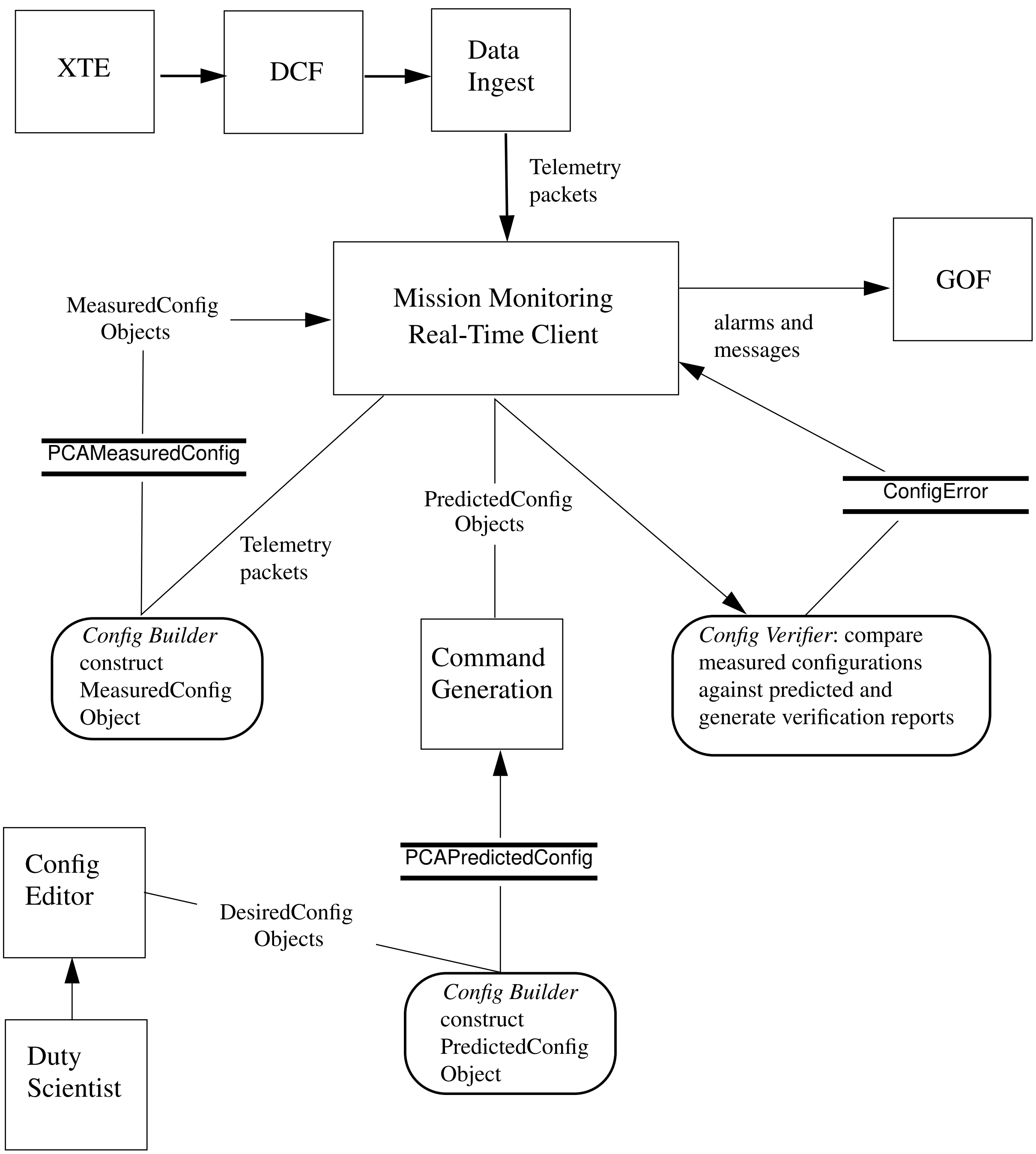
The PCA Measured Configuration software consists of two functional areas: (1) A Configuration builder and (2) a Configuration verifier. The Configuration builder will construct persistent measured configuration objects from the incoming PCU housekeeping telemetry packet. These objects are initialized to with *don't know* values then continuously updated with measured configuration values as they become available.

The Configuration verification is achieved by constantly comparing the measured configuration values against the predicted configuration values supplied by the Command Generation Subsystem. Special treatment is given to those predicted configuration parameters marked as *don't care*s. Measured configuration values that fall outside of error thresholds are reported to Operations in the form of a `VerificationReport` object containing alarms/messages with appropriate levels of severity.

6.2 Predicted Configuration

The PCA Predicted Configuration software being delivered in Build 2 would have limited functionality. It would contain a basic definition of the `PCAPredictedConfig` subclass for a limited PCA Housekeeping data.

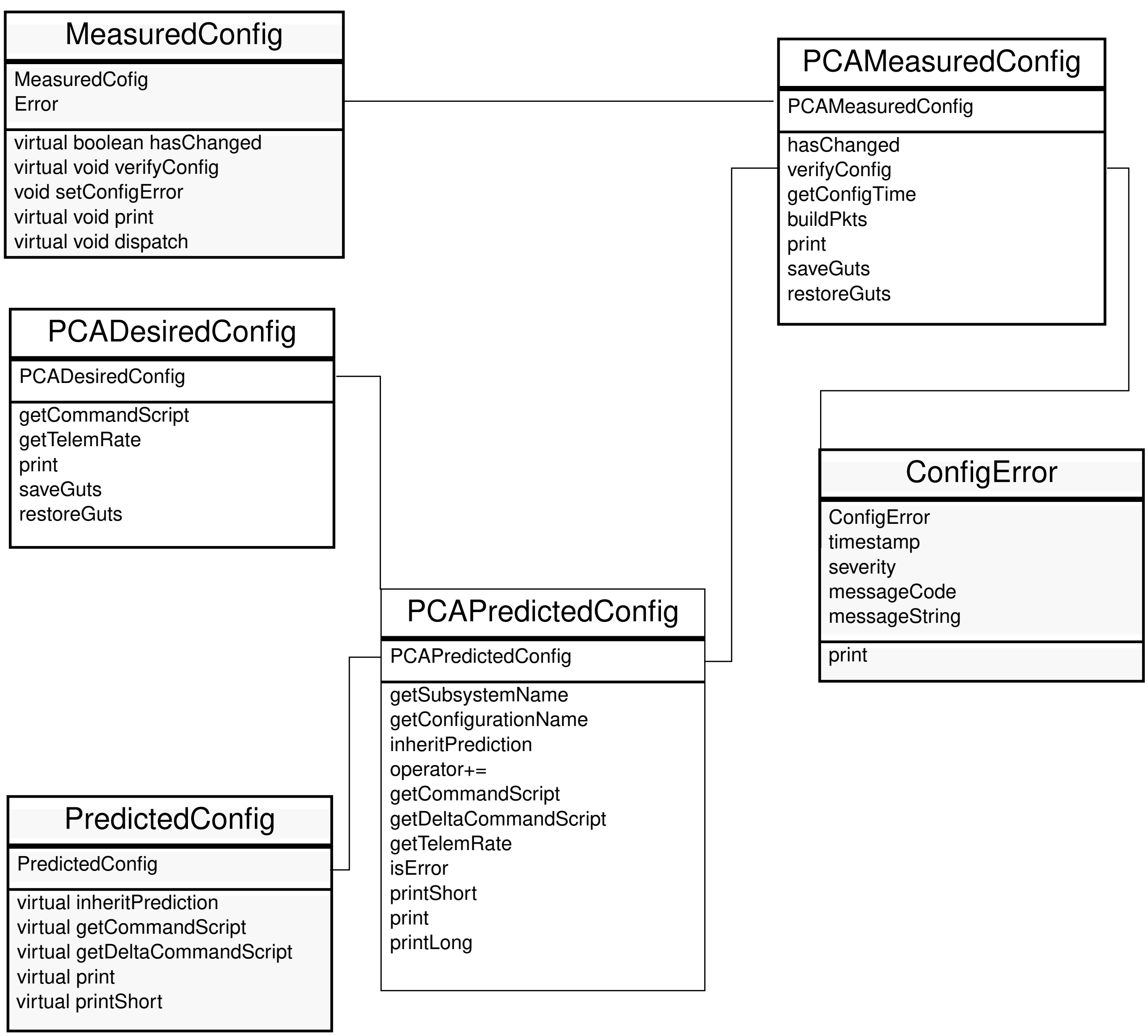
FIGURE 1. PCA Configuration Monitor: Functional flow



6.3 Class Description

Figure 2 is an object model of the configuration verification. It is quite generic. It gives a simple picture of how the configuration monitoring software is organized into objects.

FIGURE 2. PCA Configuration Monitor: Class Description



PCAMeasuredConfig

Synopsis

```
#include <PCAMeasuredConfig.h>
PCAMeasuredConfig mc;
```

Description

This is a derived subclass of the abstract base class `MeasuredConfig` for specifying the measured configurations of observatory subsystems (as opposed to predicted configurations). The SOC develops the base class and the PCA Instrument Team develops the subclass for their subsystems that inherit from this base class. The PCA Instrument Team is responsible for creating persistent measured configuration objects from the telemetry and for verifying them against predicted configurations as requested by the Mission Monitoring subsystem. The detailed mechanism for announcing the creation of these objects to clients of the ingest system is an interface internal to the SOC.

Public Constructor

`PCAMeasuredConfig ();`
This constructor creates a measured configuration object. The constructors for the derived classes set all configuration parameters to the *don't know* value.

Public Member Functions

`RWBoolean buildPackets(const PktTlm* packet);`
Builds the PCAMeasuredConfig object with the current values from the telemetry stream. When the function is invoked the configuration parameters of the PCAMeasuredConfig object are updated from their current values as they become available. It returns TRUE if the configuration is complete; FALSE otherwise. The `buildPackets` method is called for every measured configuration.

`long getConfigTime () const;`
Returns the time associated with the measured configuration object, which is then used to request the appropriate predicted configuration(s) for comparison.

`RWBoolean hasChanged (MeasuredConfig* prev) const;`
Returns TRUE if the it detects that this configuration differs *significantly* from the previous measured configuration, and FALSE otherwise. The Instrument Teams define what is a significant change. One consequence of a significant change is that this PCAMeasuredConfig object is written to the configuration history file. Non-significant changes are not recorded.

`VerificationReport* verifyConfig (const PredictedConfig* predicted) const;`
Compares this PCAMeasuredConfig with the given PredictedConfig and returns a VerificationReport object describing the comparison results. If the comparison reveals no *significant* differences, this method may return a null pointer. The Instrument Teams define the term "significant". The PredictedConfig will be an object previously created during command generation. The Command Generation subsystem provides a method to be used by Mission Monitoring to retrieve the PredictedConfig object from the timeline of predictions.

```
void print(ostream& ostr) const;
```

Displays the current PCAMeasuredConfig object.

PCAPredictedConfig

```
#include <PCAPredictedConfig.h>
PCAPredictedConfig pc;
```

Description

This is a subclass for specifying the *predicted* configurations of observatory subsystems and for generating command scripts. The SOC develops this base class and the Instrument Teams develop subclasses for their subsystems that inherit from this base class and implement the virtual methods described below.

This class differs from the PCADesiredConfig class in that it records a predicted value for as many configuration parameters as possible, inheriting values from previous predictions. Parameters may have *don't know* values if predictions are unavailable. (Be careful to distinguish this from the *don't care* value that appears in a desired configuration.)

Each PCAPredictedConfig object is associated with a PCADesiredConfig object. It necessarily makes a copy of all information in that DesiredConfig. One possible implementation simply has a PCADesiredConfig object as a private data member of the PCAPredictedConfig.

Public Constructor

`PCAPredictedConfig (const PCADesiredConfig& dc)`
This constructor creates a new predicted configuration object given a desired configuration.

All subclasses of PCAPredictedConfig must invoke this constructor. They copy all configuration values and *don't care* flags from the `DesiredConfig` and set a *don't know* value for all *don't care* configuration parameters.

Public Member Functions

`const char* getConfigurationName() const;`
Returns a pointer the configuration name originally specified in the `DesiredConfig` object.

`const char* getDescription () const;`
Returns a pointer to the descriptive text originally specified in the `DesiredConfig` object.

Virtual Member Functions

`SubsystemId getSubsystemId() const;`
Returns the subsystem identification, an enum defined as follows.

```
enum SubsystemId {PCA, HEXTE, ASM, EDS, PRAMA, PRAMB, EA1,
EA2, EA3, EA4, EA5, EA6, EA7, EA8};
```

```
void inheritPrediction (const PCAPredictedConfig&
priorConfiguration)
```
This method copies values from the `priorConfiguration` into this one. Only the parameters with *don't care* settings are updated; parameters that were specified in the constructor's `PCADesiredConfig` are unchanged. This operation can be repeated with a different `priorConfiguration` argument; it does not disturb *don't care*

flags or the values of non-*don't care* parameters. A second `inheritPrediction` operation will result in a PCAPredictedConfig object exactly as if the first operation had not occurred.

If this method encounters a problem and cannot inherit the `priorConfiguration`, it constructs a `SOCErrObject` to be returned by the `isError` method, issues a message on standard error, and retains all affected configuration parameters as *don't cares*.

PCAPredictedConfig`* deriveConfig ();`
This method returns a pointer to a new PCAPredictedConfig to be inserted onto the predicted time-line immediately before the current entry. This method is called repeatedly until it returna a null pointer.[1]

`CommandScript* getDeltaCommandScript() const;`
This method returns the command script that puts the subsystem into this predicted configuration. Only the parameters *not* predicted to have their desired value are commanded. (Compare this with the `getCommandScript` method.)

If this method cannot produce a valid CommandScript, it constructs a `SOCErrObject` to be (1) returned by the `isError` method and (2) inserted into the CommandScript itself using its `setErrObject` method. This method also issues an error message on standard error. All configuration parameters that would have been set by the `CommandScript` become *don't knows*.

`CommandScript* getCommandScript() const;`
Returns the CommandScript that puts the subsystem into this predicted configuration. No configuration prediction is assumed; *all* configuration parameters are commanded, except, of course, those with *don't care* values. (Compare this with the `getDeltaCommandScript` method.)

If this method cannot produce a valid `CommandScript`, it constructs a `SOCErrObject` to be (1) returned by the `isError` method and (2) inserted into the `CommandScript` itself using its `setErrObject` method. This method also issues an error message on standard error. All configuration parameters that would have been set by the `CommandScript` become *don't knows*.

`TelemRate* getTelemRate(const Source& source) const;`
Returns the average telemetry rate of the subsystem in the current configuration.

`SOCErrObject* isError() const;`
Returns the current error object for this class,[2] or zero if the preceding operation did not incur an error.

Virtual Print Functions

`void printShort (ostream& ostr) const;`
Prints a one-line identification of this configuration on the output stream.

1. This method is intended for use primarily by the EDS programmers.

2. `SOCErrObject` inherits from `RWErrObject` of the Rogue Wave tools.h++ class library.

```
void print (ostream& ostr) const;
```

Prints a description of this configuration on the output stream, at least as detailed as the one given by `printShort`.

```
void printLong (ostream& ostr) const;
```

Prints a description of this configuration on the output stream, at least as detailed as the one given by `print`. It includes all the configuration values that would be of primary interest to the Duty Scientist while reviewing observation schedules or during initial scans of incoming science data.

7.0 Interfaces

The PCA Measured Configuration described here will interact mainly with the Mission Monitoring subsystem provided in the SOC. It will also indirectly provide input to the Command Generation subsystem when it is necessary to build a desired configuration based on the last known state of the instrument.

The PCA Predicted Configuration software will mainly interact with the Command Generation subsystem provided in the SOC. The subclass implementation of the SOC base class will define and track all instrument configurations and generate intrument commands. It will also indirectly provide input to the Mission Monitoring subsystem for verifying with measured configurations and will also provide input to the Configuration Editor for building a desired configuration.

8.0 Abbreviations and Acronyms

DCF	Data Capture Facility
EOR	Execute On Receipt
GOF	Guest Observers Facility
IAD	Interface Agreement Document
ICD	Interface Control Document
PCA	Proportional Counter Array
PCU	Proportional Counter Unit
RIF	Remote InterFace Unit
SOC	Science Operations Center
XTE	X-ray Timing Explorer

9.0 Glossary

10.0 Notes